General Modeling Principles: Building Blocks of Analysis Patterns

PA116 – L2

(c) Zdenko Staníček, Sept 2010

INVESTMENTS IN EDUCATION DEVELOPMENT

Content

- Data Polymorphism
- Special Constructs (Analytic patterns)
- Analytic Pattern *Accountability*

According to:
Lubor Sesera: presentations
papers (“google it, please”)
books (Application Architectures
of SW Systems – in Slovak)

Data Polymorphism - topics

- What is data polymorphism?
- What are reasons for using data polymorphism?
- Solving data polymorphism by using "*glasses*"
- Solving data polymorphism by using attribute as isolated entity

  (see Special Constructs or Building Blocks in DM)

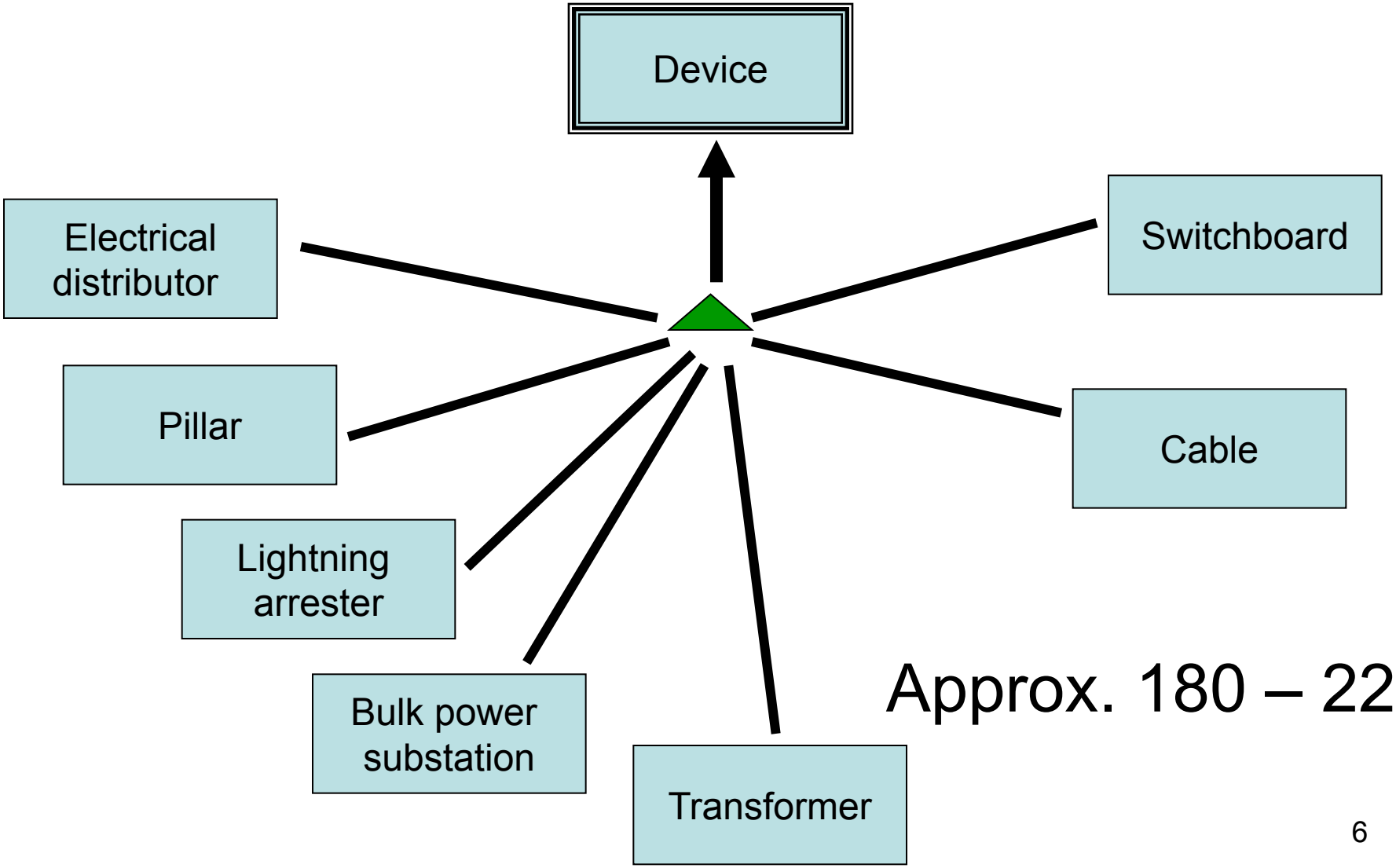
What is ***data*** polymorphism?

- Type of data processed by a program is not known in compilation time, but only in the time of program running.
- Function represented by program is computed in the same way for different types of its arguments
- Typical example is function “Sum”
- … along with this the data processing lies in
  - ***permanent storage in DB***,
  - searching, actualizing, ...

Reasons for utilization of data polymorphism

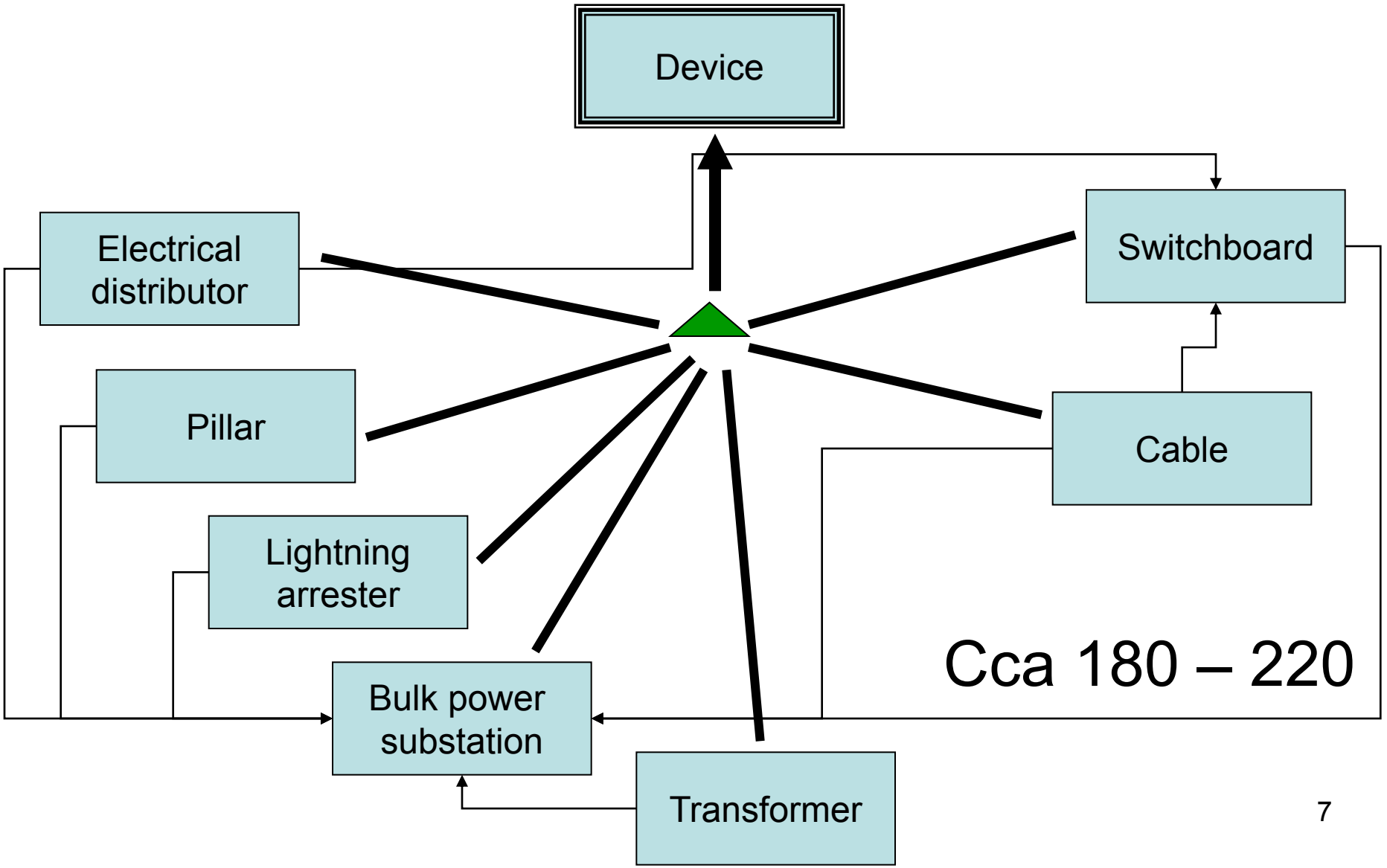
- Example: Component “Technical Equipment Register and Maintenance” in Power distribution plant
  - What *Item* is worth *to be filed as self-reliant entity*
  - Would we like to see “*Transformer”* and “*Electrical Distributor”, (“insulator”)* or only
  - “*Device*”
- Complexity of application when data polymorphism is not used
- Dependence of application on current state of users’ thinking about their business needs
- Sliding on the Identity-axe (what has to be distinguished and what hasn’t)

Supertype--Subtype, Identity of types

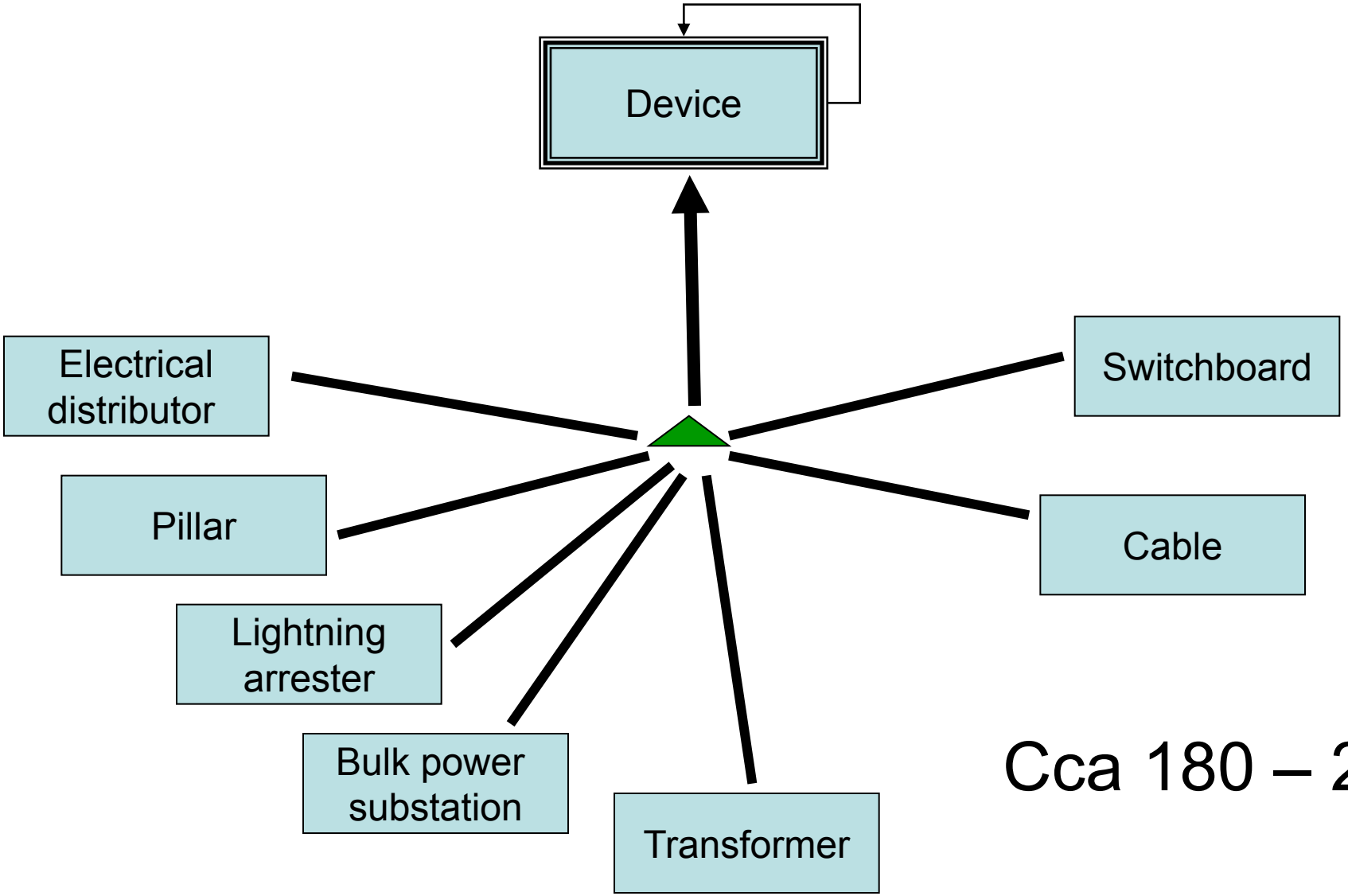


Approx. 180 – 220

Connections between Subtypes



Connections between Supertypes

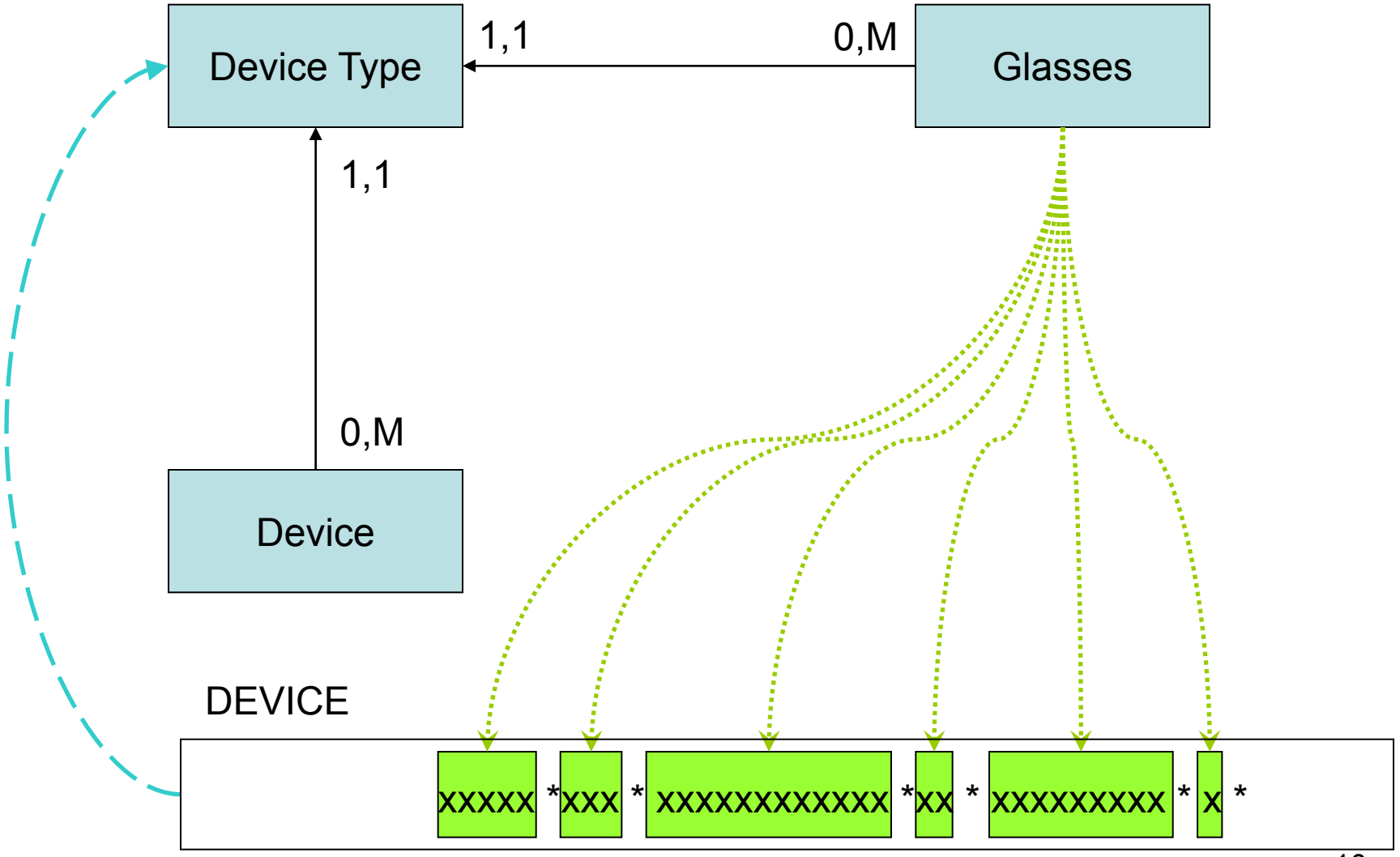


Cca 180 – 220

Solving Data Polymorphism using "*glasses*"

- Non-interpreted storage place in *Device* records
- Each *Device* instance is of exactly one type from *Device Type*
- Within the run-time the current "type" of the "non-interpreted storage" of a given *Device* instance is interpreted according to assigned "device type"

Using *glases*



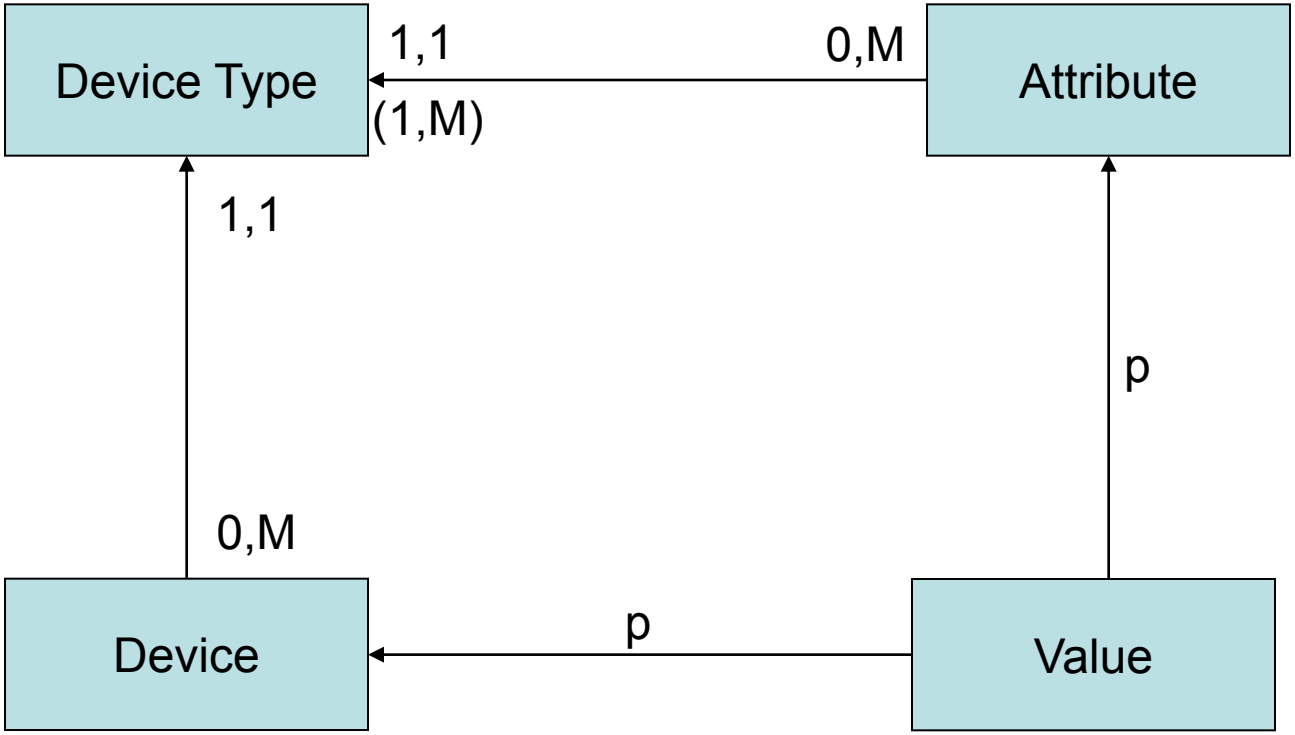
Solving Data Polymorphism using "*glasses*" -- discussion

- Used in several real-life examples (case study "Building Data Model", IS "White Butterfly", …)
- Discussion:
  - What are the advantages?
  - What kind of problems this solution brings?
  - Is it easy to maintain?
  - Computational complexity?
  - What about the fact that SW company has its product at 100 (at 1000, ...) customers?

Solving Data Polymorphism by using attribute as isolated entity

- Who will find out how to do it?
- It is one of the basic *analytic patterns*

Using attribute as isolated entity



(Value) of given (#Attribute) for given (#Device) / 0,1:0,M

Attribute as isolated entity

- Discussion:
  - What are advantages?
  - What kind of problems this solution brings?
  - Is it easy to maintain?
  - Computational complexity?
  - Comparison with *Glases*

Special Constructions (Analytical patterns)

- Are there any other general solutions except the Data Polymorphism?
- Lubor Šešera: paper at DATASEM
- ... and the book *Data Modeling in Examples*
- *… and now: Application Architectures of SW Systems*
- M. Fowler: *Analysis Patterns: Reusable Object Models.*

Reasons for usage

- Example: Component “Technical Equipment Register and Maintenance”
- Complexity of application when special constructs are not used
- Dependence of application on current state of users’ thinking about their business needs
- Higher solidity of the construction when proven prefabricated elements are used

Conceptual modeling using patterns - principles

- Principles of normalization
- Principles of abstraction
- Principles of flexibility

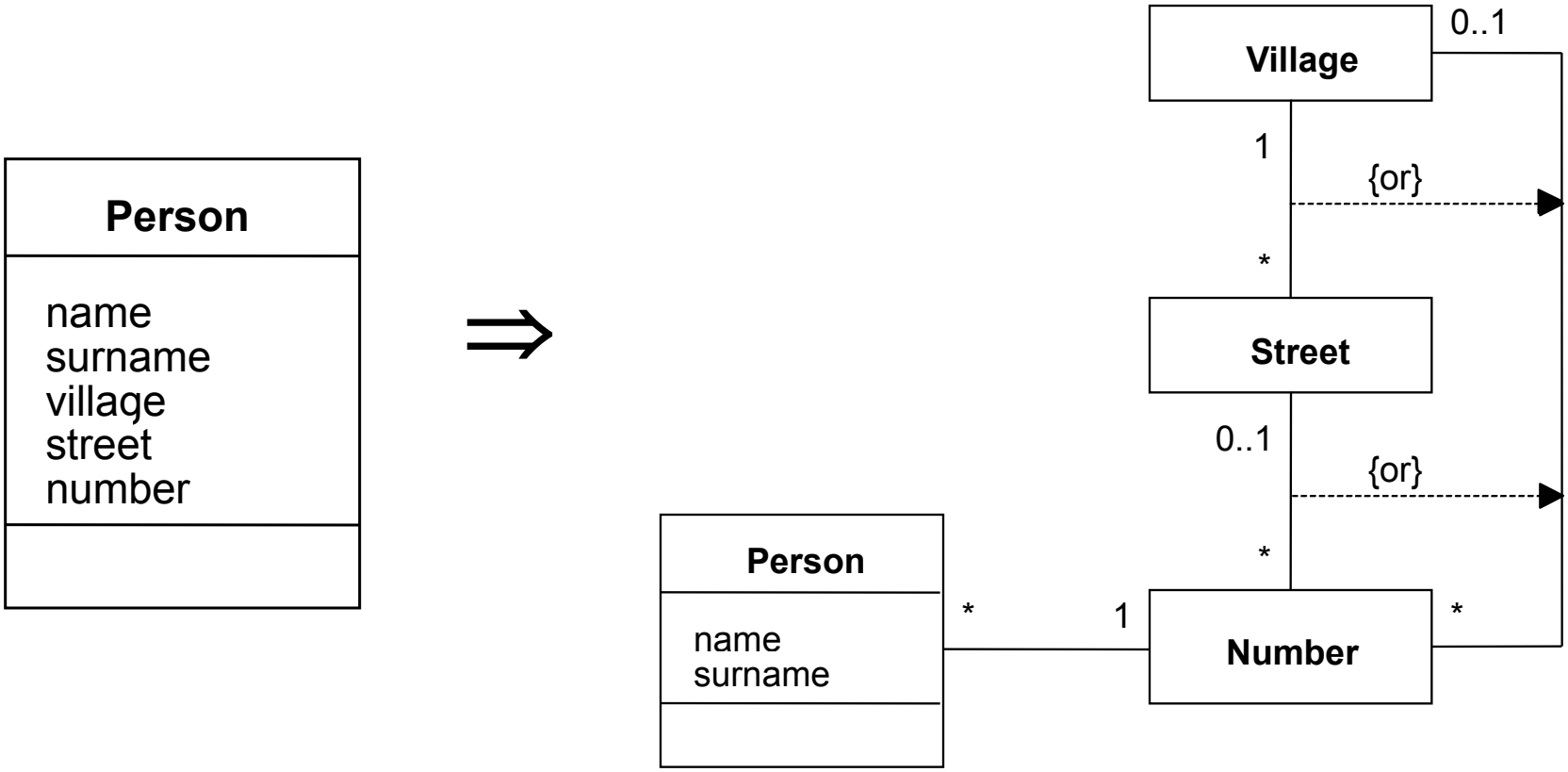
- See Šešera: Data Modeling in Examples

  Following examples and figures are used from authors speach (with author's kind permition) L. Šešera: General Data Modeling Principles: Building Blocks of Analysis Patterns. DATASEM 1999, which preceded cited book

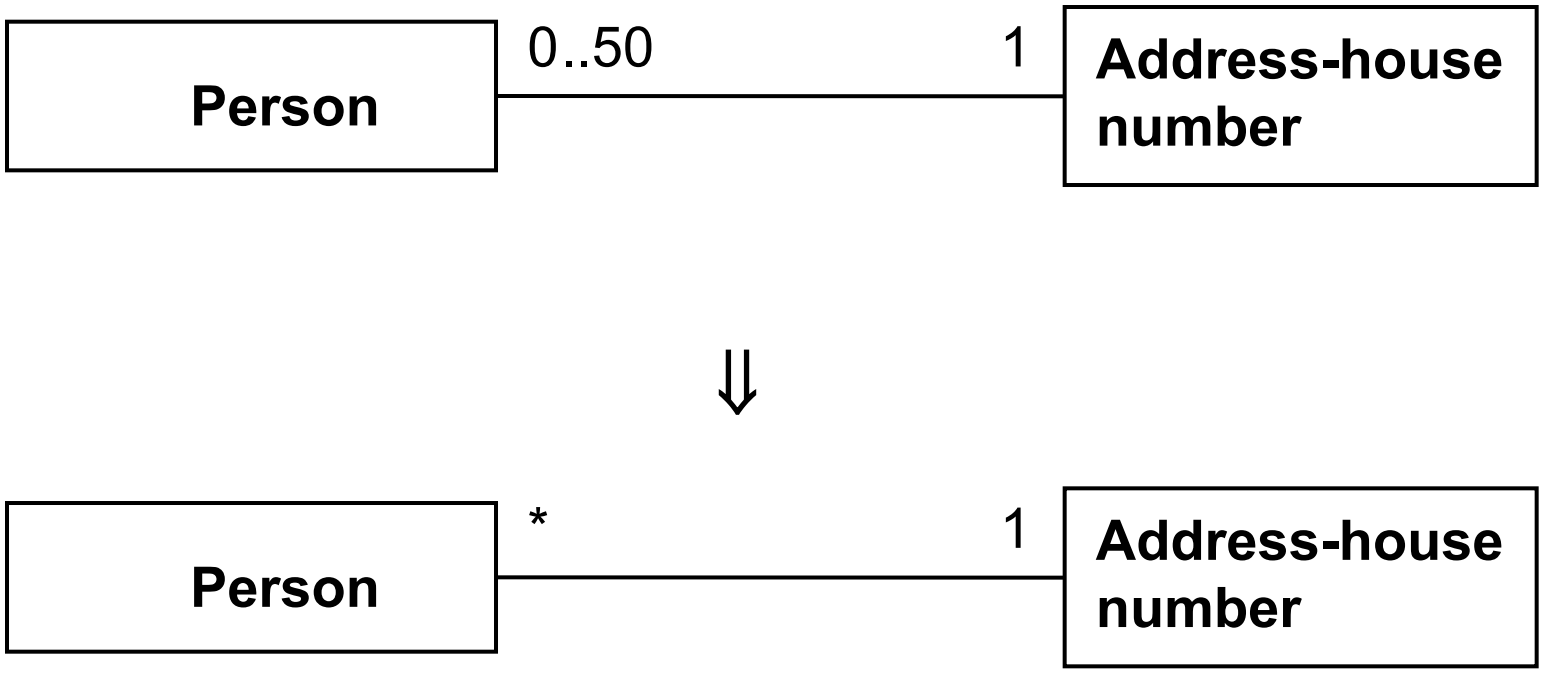
Principles of normalization

- Uniqueness of occurrence of data item
  - from not normalized entity towards to construction of normalized entities
- Multiplicity
  - reduce a multiplicity of objects in relation only to:
    - **zero**
    - **one**
    - **infinity**

Every fact only in one place



Actual cardinalities transform to general ones

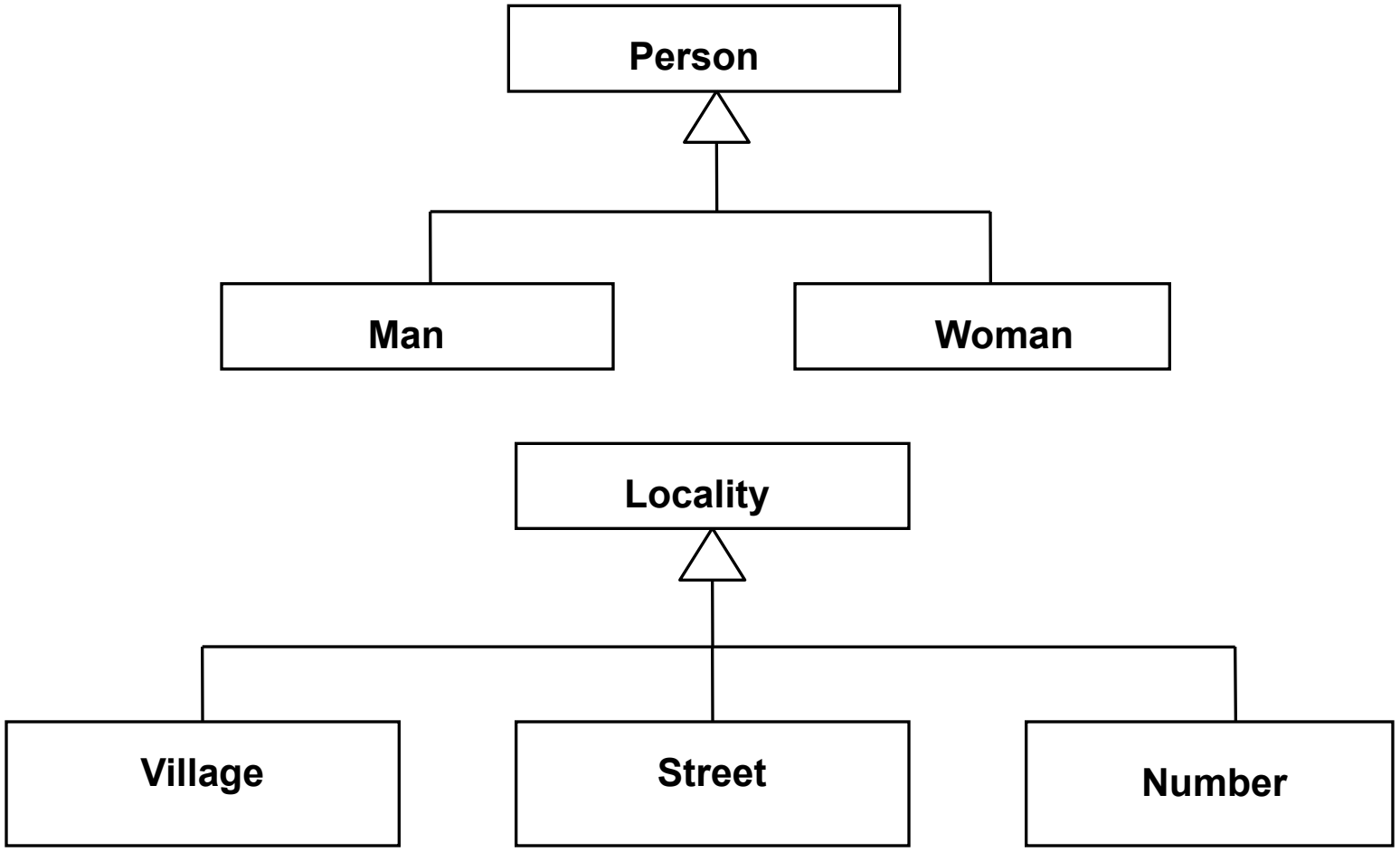


Principles of abstraction

- Abstraction in itself
  - Generalization
  - Constructing types
  - Substitution
- Aggregation
- Categorization

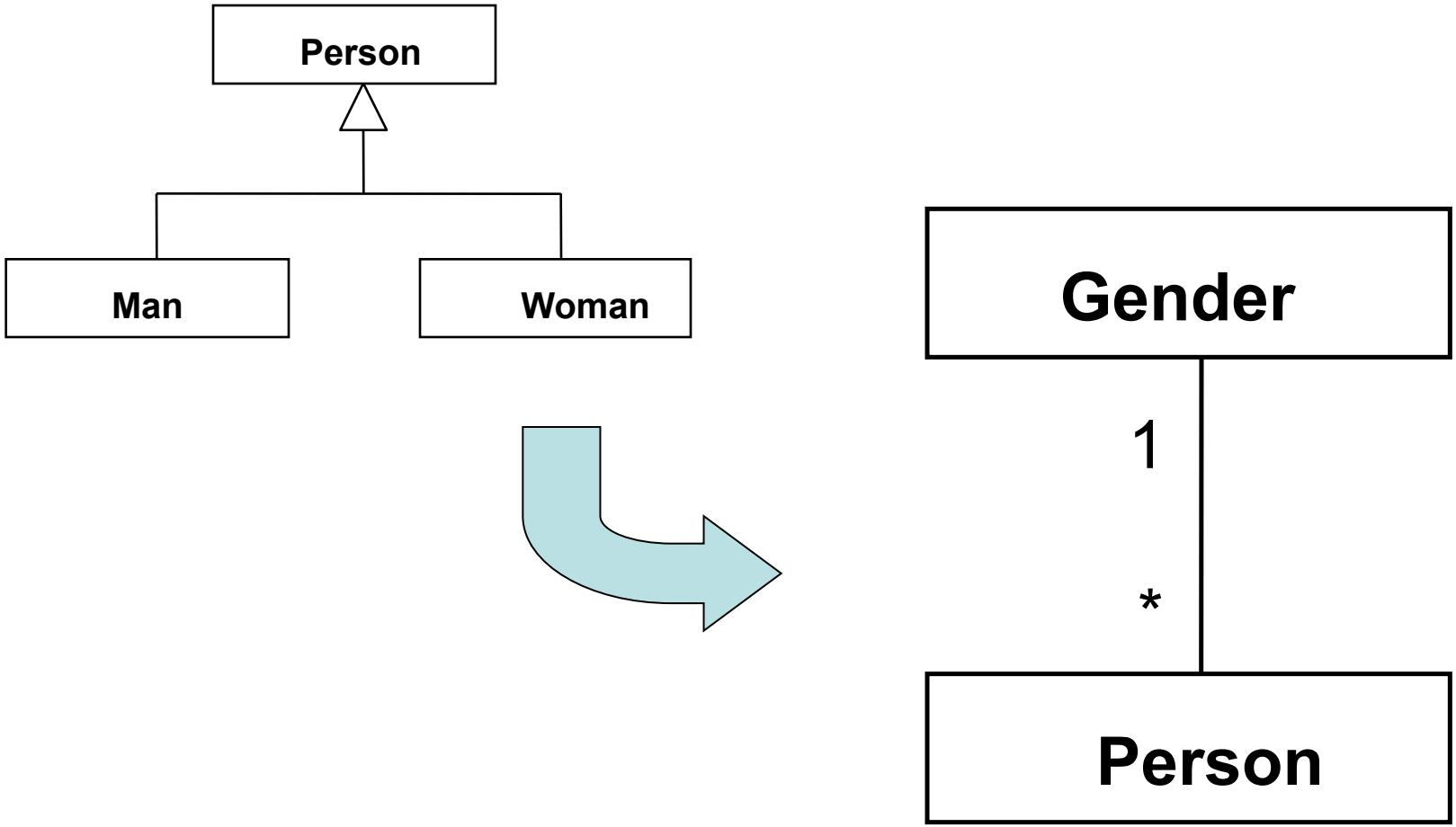
Generalization

relation of inheritance, some inherited characteristics could be overridden in subtype



Constructing types

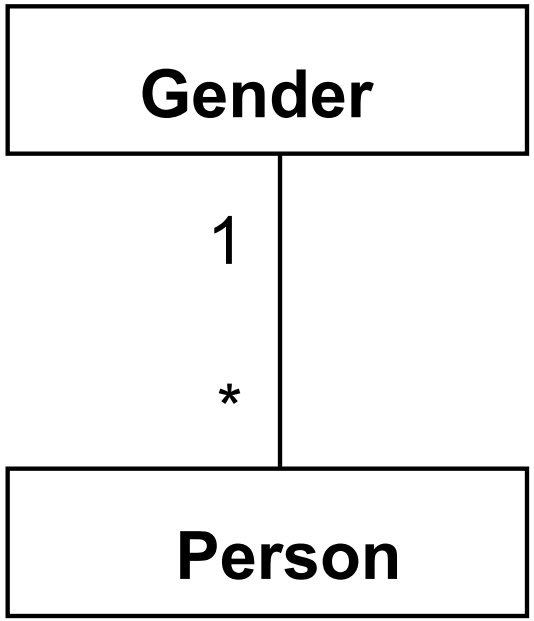
type as a standalone entity



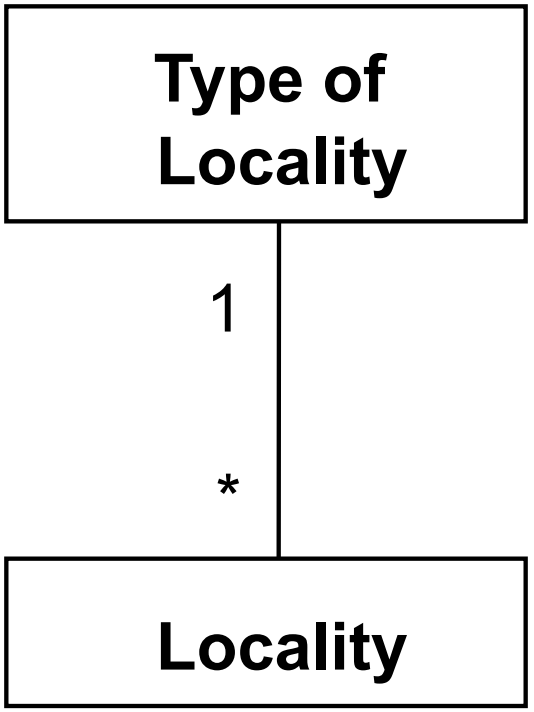
Constructing types (2)

type as a standalone entity

According to this pattern:

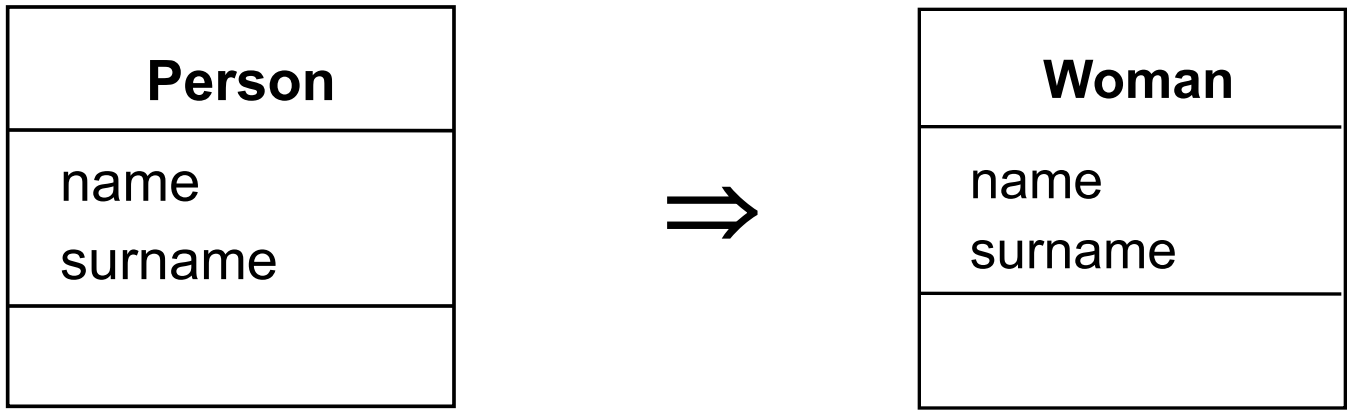


… we create the following construction:



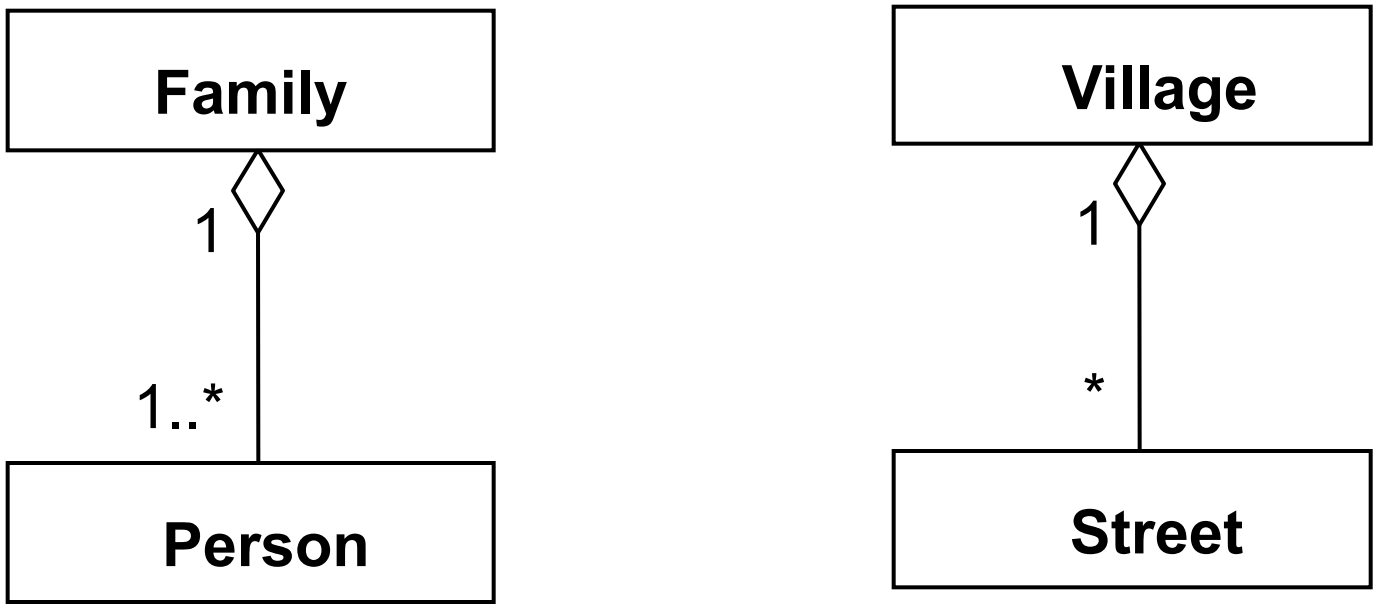
Substitution

special entity is set instead of general entity - opposite of generalization



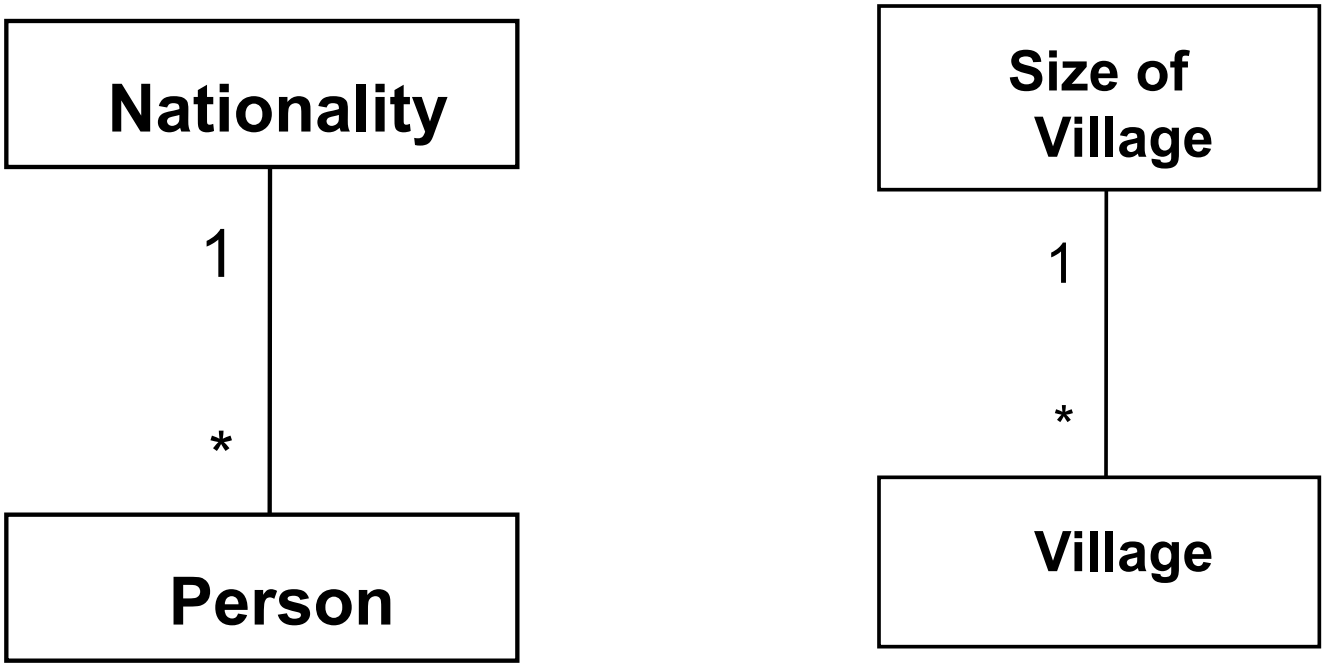
Aggregation

grouping parts into wholes; it supports constructing more levels of abstraction

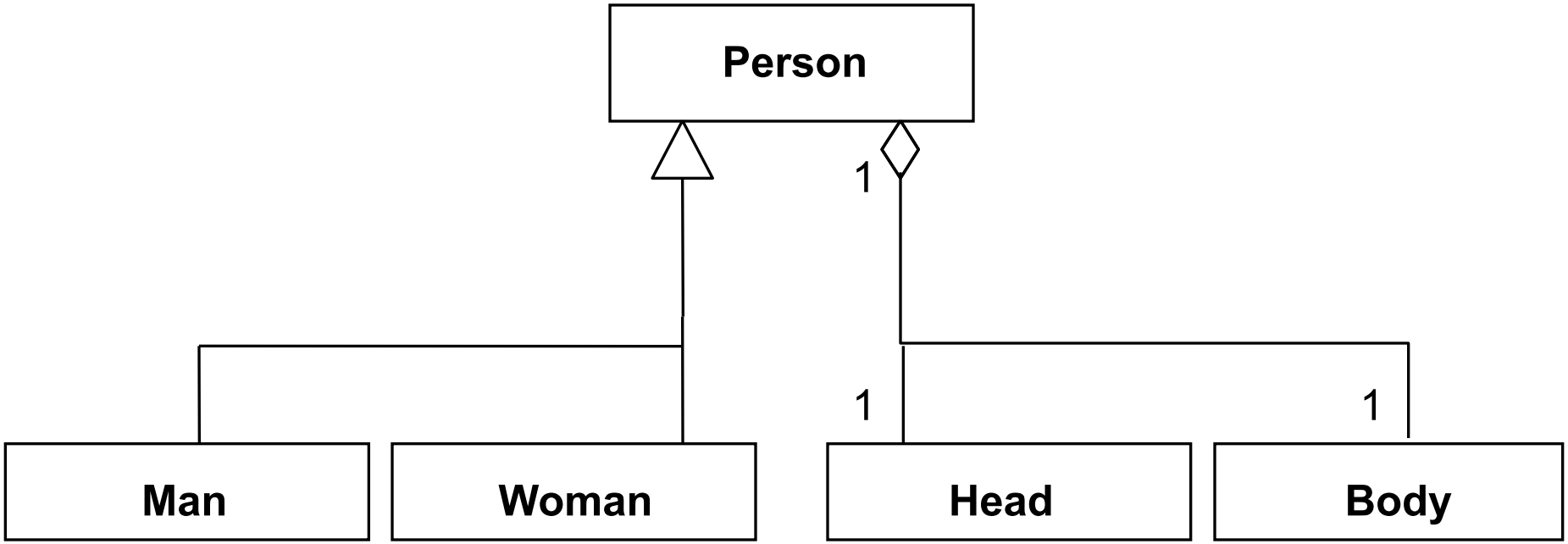


Categorization

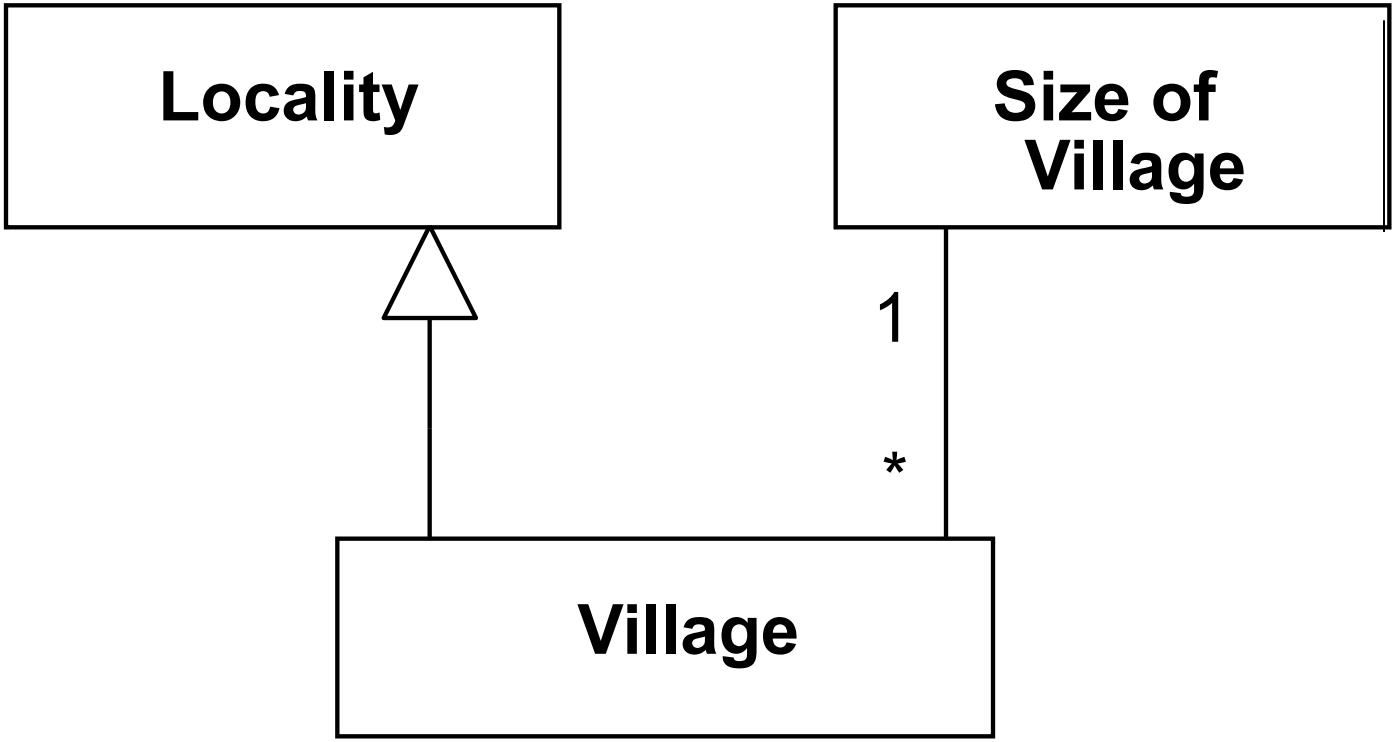
grouping instances into sets; it does not subscribe attributes to grouped instances (contrary to constructing types)



Abstraction (Generalization) vs. Aggregation



Abstraction (Generalization) vs Categorization

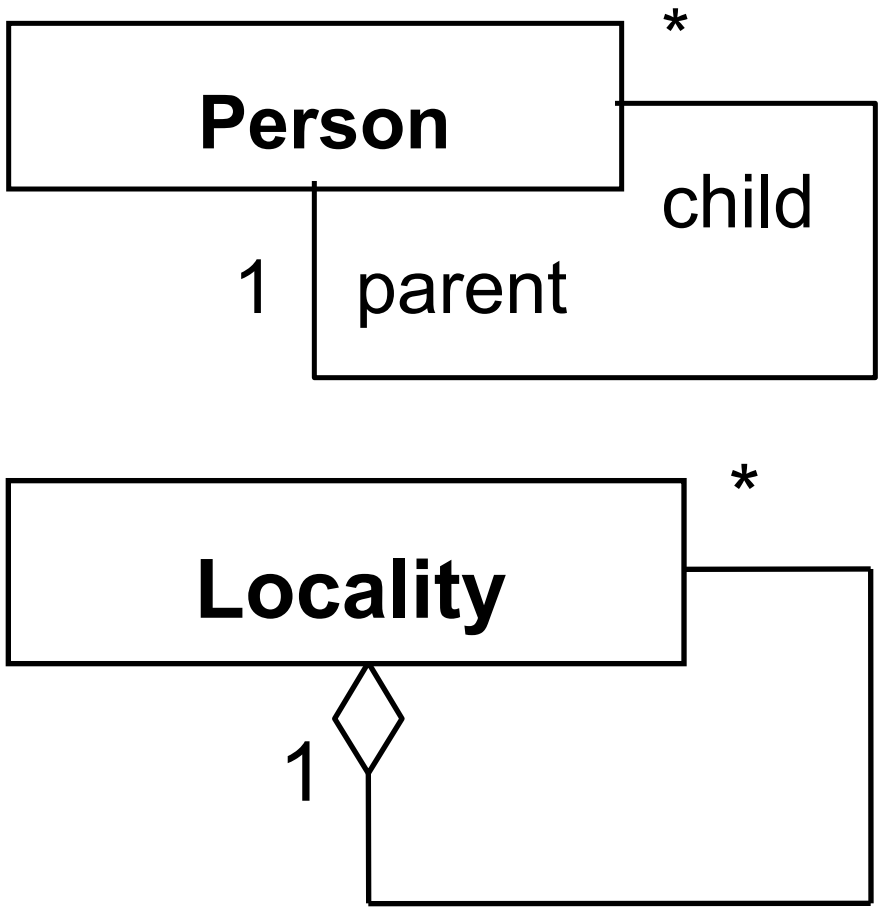


Principles of Flexibility

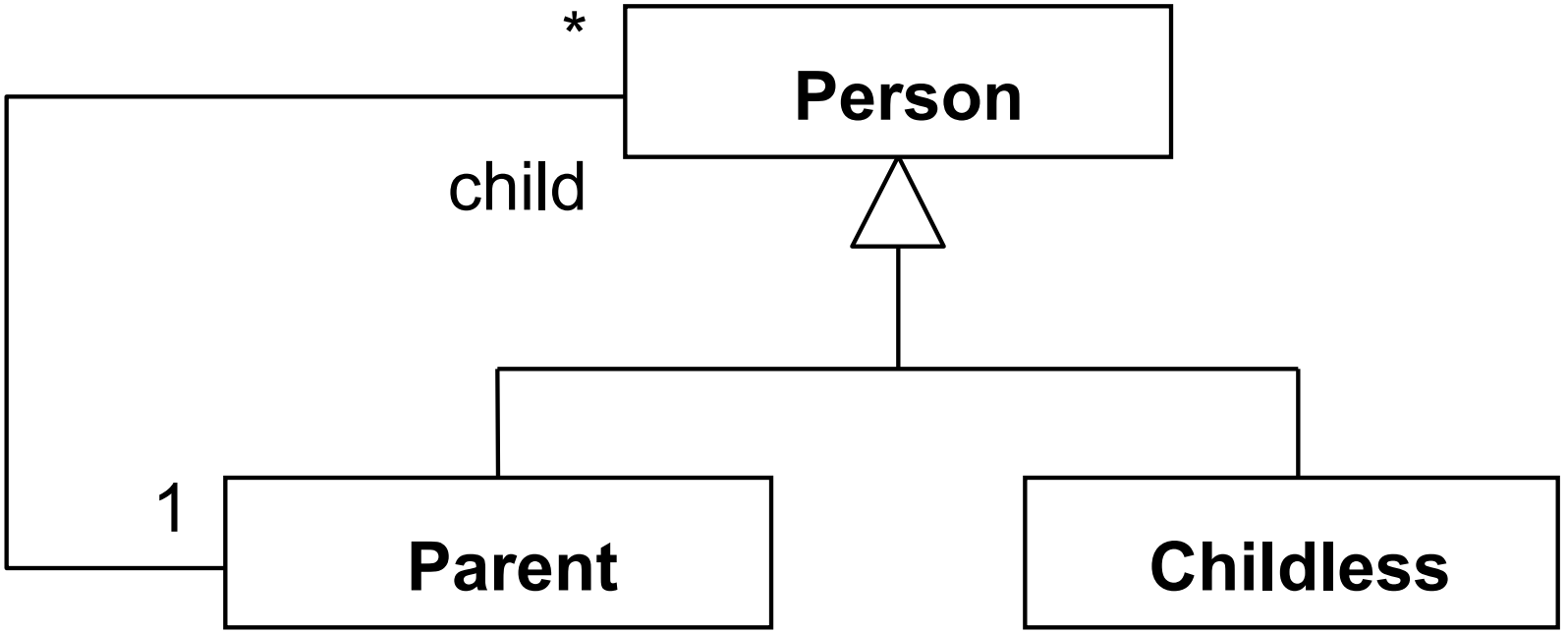
- Recursion
  - direct
  - indirect
- Abstraction of relationship
- Abstraction of attributes

Direct recursion

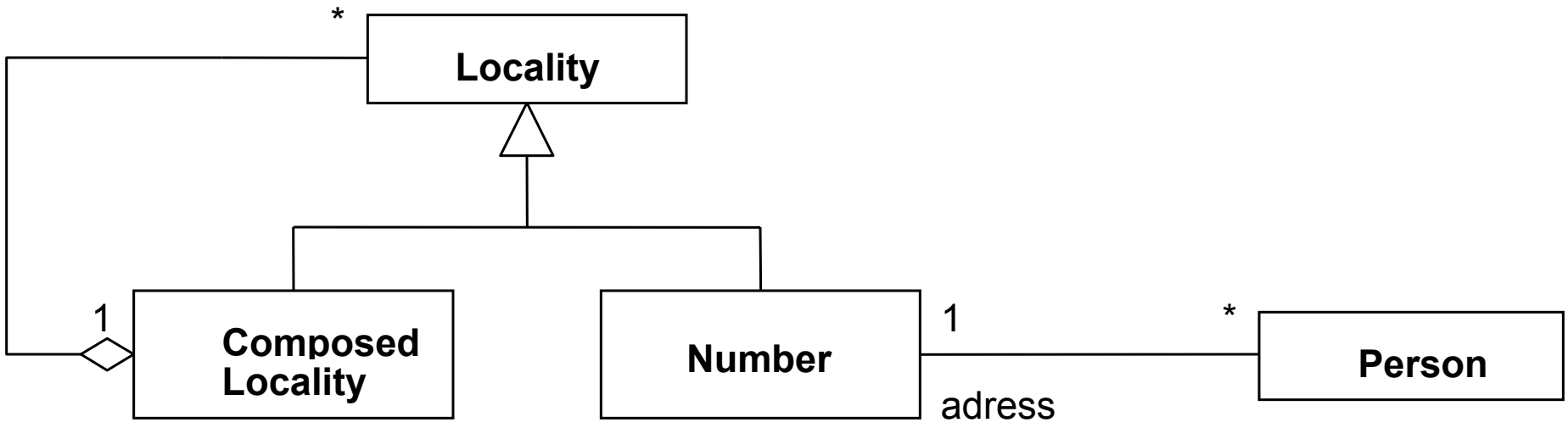
semantic relationship and special relationship (aggregation)



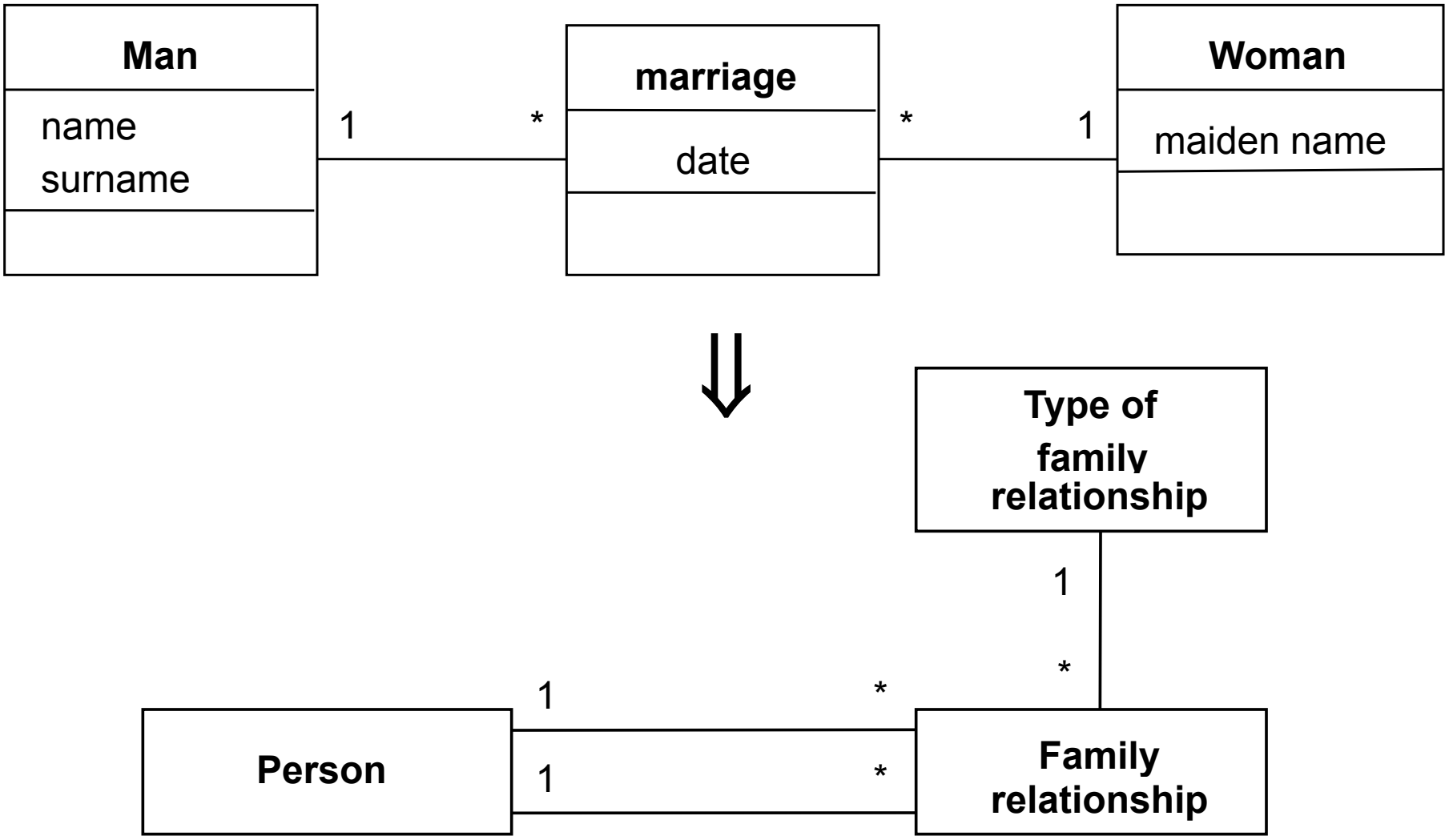
Indirect recursion (1)



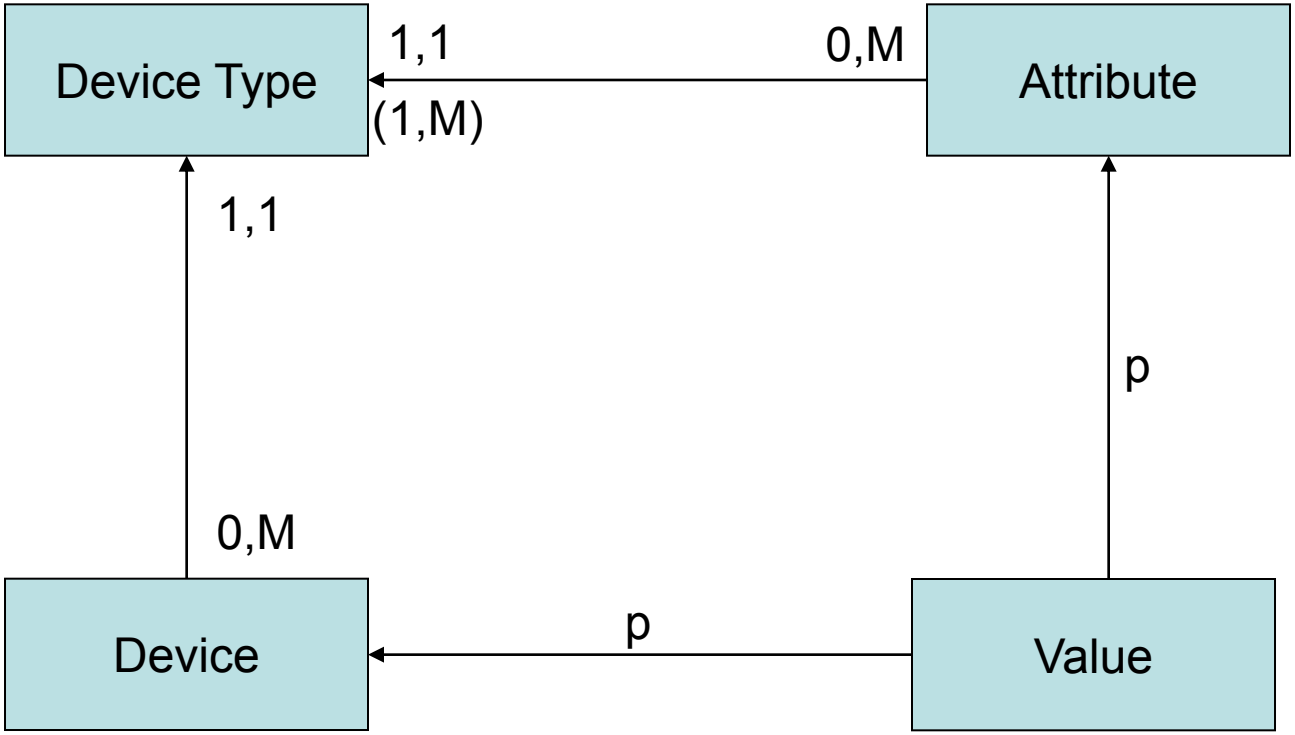
Indirect recursion (2)



Abstraction of relationships



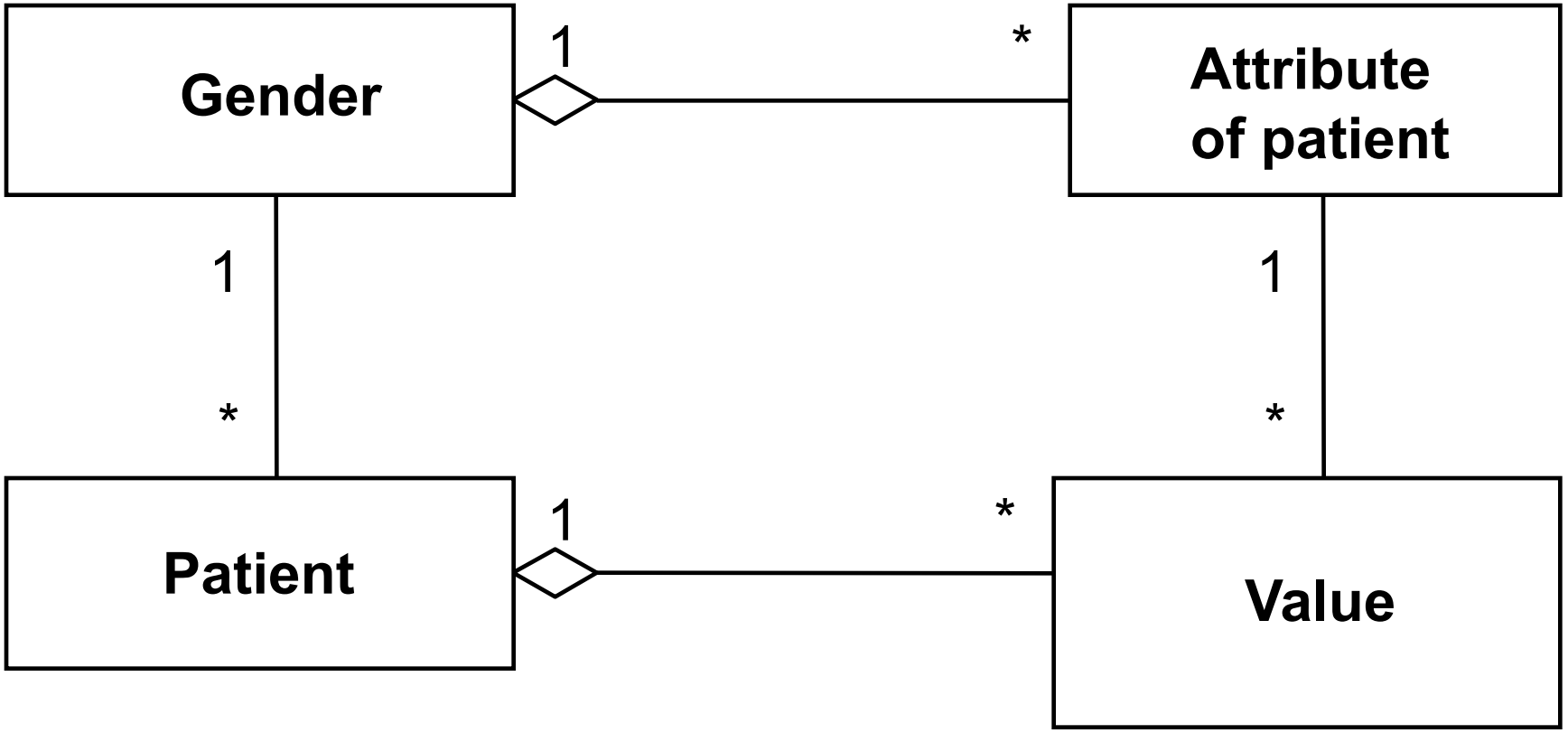
Abstraction of attributes (1)



(Value) of given (#Attribute) for given (#Device) / 0,1:0,M

Abstraction of attributes (2)

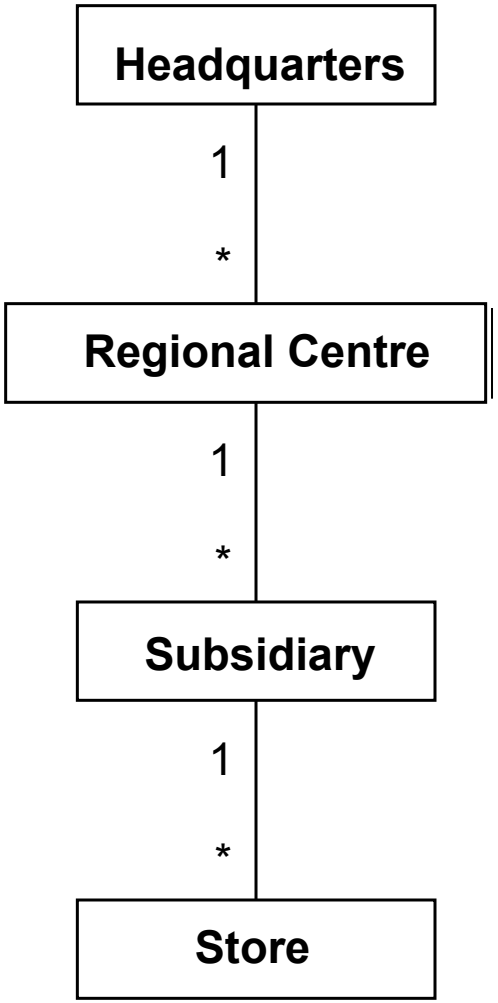
not only by using types, but using aggregations



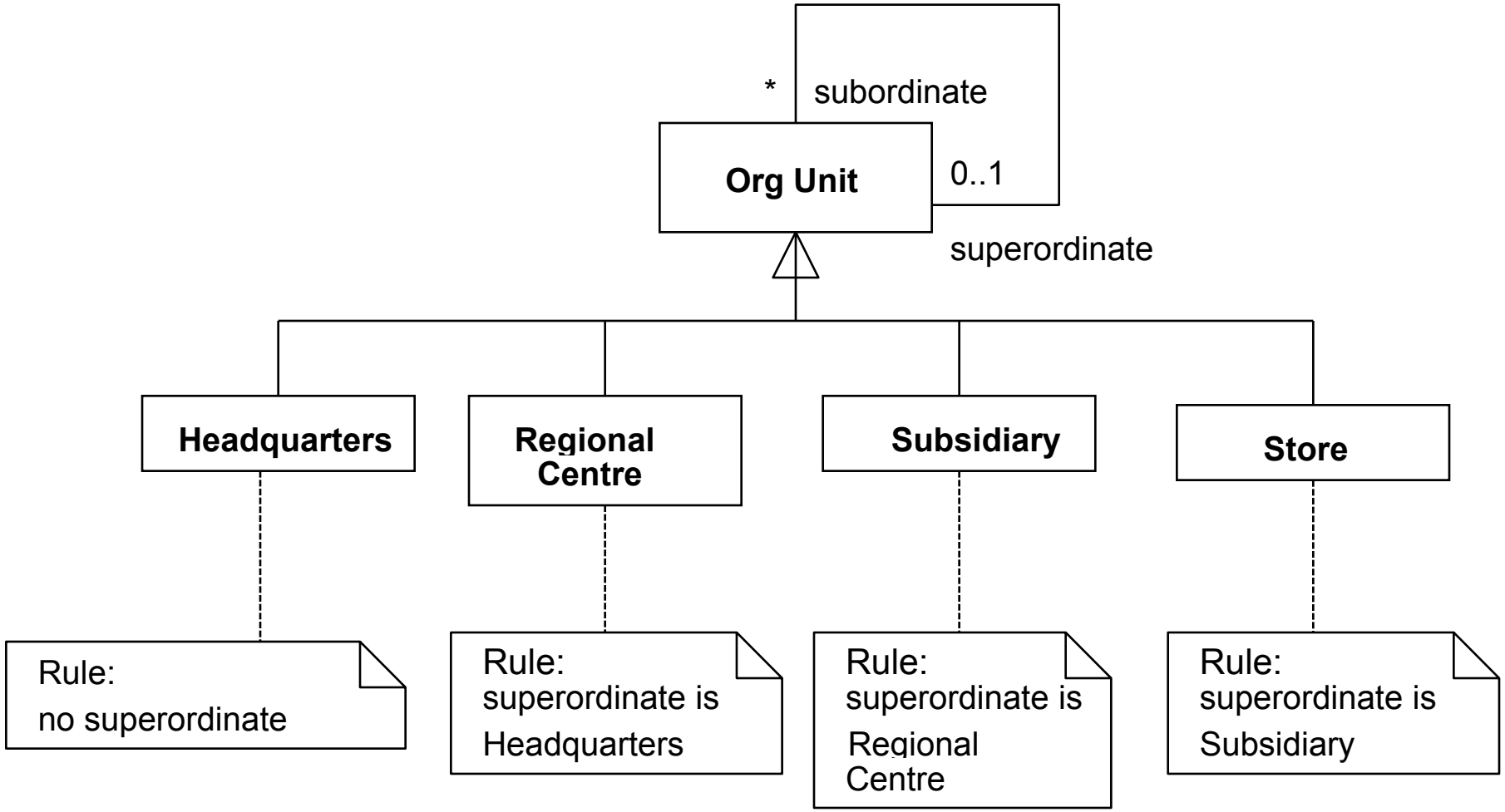
Pattern *Accountability*
as an example of complex patterns

- taken from L. Šesera's lecture at DATASEM 1999
- by M. Fowler
- Organizational structure of big organization
- Problem of changing the model when the organization structure changes
- Problem of more existing hierarchies at the same time
- Abstractions: OrgUnit → Participant; OrgRelationship→ (any) Accountability
- Separation of general knowledge from operational level; implementation of general *scope* of accountability and allowing multiple inheritance

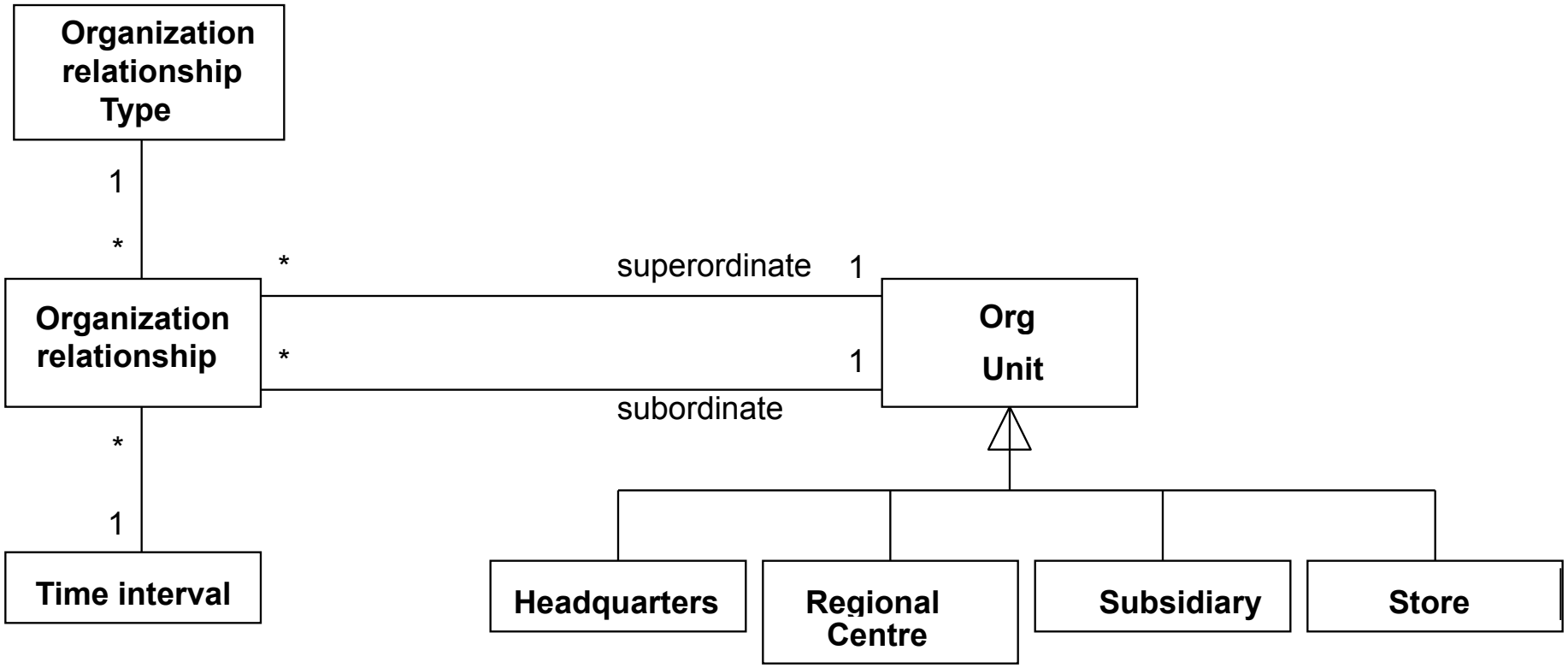
Organizational structure of big organization



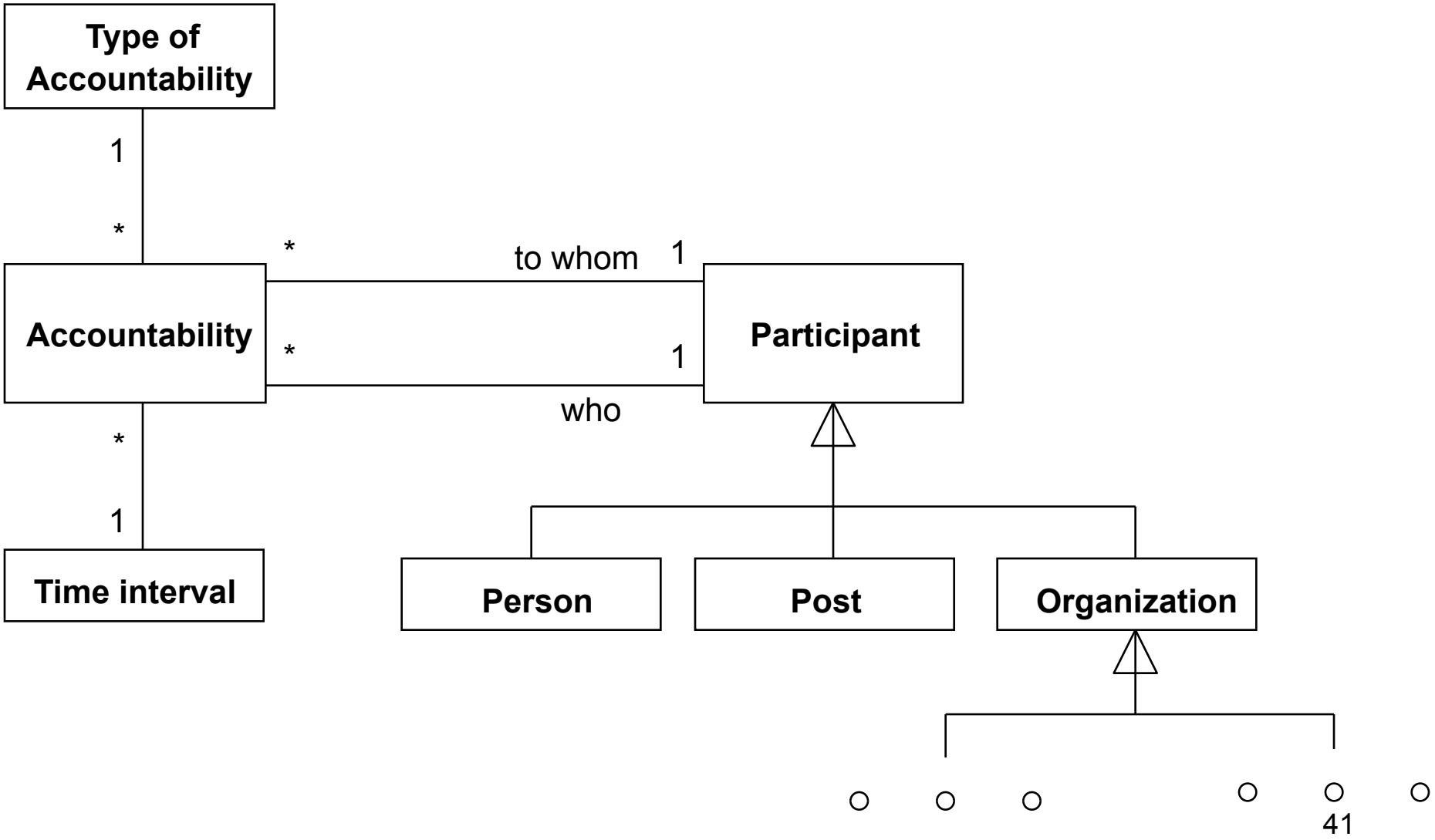
Solving Problem of changing the model when changing the org. structure



Solving Problem of more existing hierarchies at the same time



Abstraction: OrgUnit → Participant
OrgRelationship→ (any) Accountability

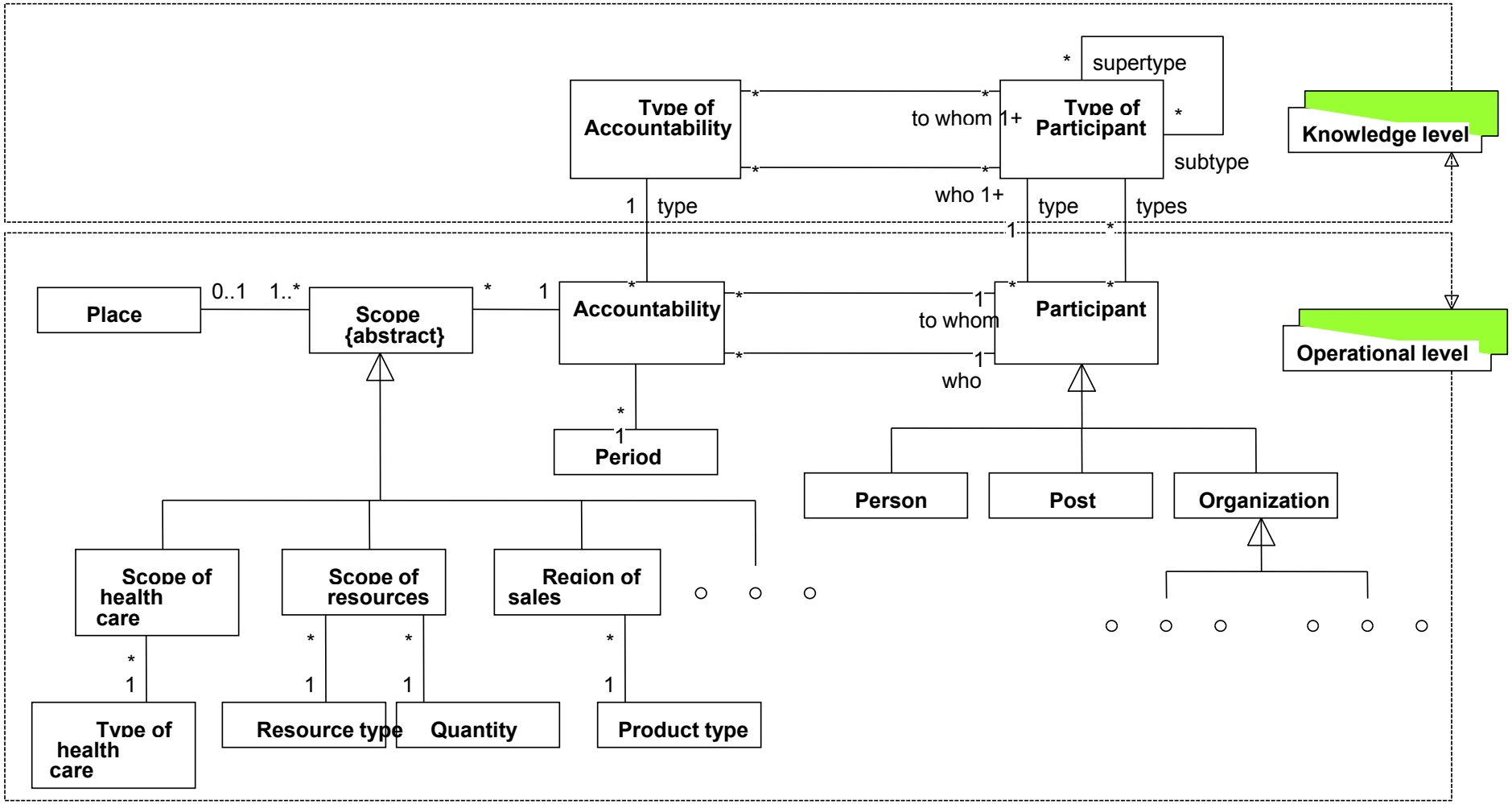


Generalization up to analytic pattern *Accountability*

- Separate general knowledge of the world from operational level on which we keep track of particular states-of-the-world
- Implement general *scope* of accountability
- Allow multiple inheritance (we inherit from mother and also from father; subtype can have more supertypes)
- Subtypes of relational entity *Scope* and explanation what is the scope for (subtype entities of the *Scope*)

Analytic pattern Accountability

by Lubor Šešera, DATASEM'99



Discussion

- Fear of special constructs
- Quo vadis SW development?
- Will we implement solutions after domain and situation analysis in the future? Or will we teach a Service System what to do?
- What customers want *today*?
- What will they want *tomorrow*?
- … and what they really need?